CSE 543 - Computer Security (Fall 2006)

Lecture 24 - Virtual machine security
November 28, 2006
URL: http://www.cse.psu.edu/~tjaeger/cse543-f06/
Operating System Quandary

- Recall Saltzer-Schroeder
  - Q: What is the primary goal of system security?
- OS enables multiple users/programs to share resources on a physical device
  - Access control policies of OS become complex
  - We’ll see via SELinux
- What are we to do?
Virtual Machines

• Instead of using system software to enable sharing, use system software to enable *isolation*

• Virtualization
  - “a technique for hiding the physical characteristics of computing resources from the way in which others systems, applications, and end users interact with those resources”

• Virtual Machines
  - Single physical resource can appear as multiple logical resources
Virtual Machine Architectures

- **Full system simulation**
  - CPU can be simulated

- **Paravirtualization (Xen)**
  - VM has a special API
  - Requires OS changes

- **Native virtualization (VMWare)**
  - Simulate enough HW to run OS
  - OS is for same CPU

- **Application virtualization (JVM)**
  - Application API
Virtual Machine Types

- **Type I**
  - Lowest layer of software is VMM
  - E.g., Xen, VAX VMM, etc.

- **Type II**
  - Runs on a host operating system
  - E.g., VMWare, JVM, etc.

- Q: What are the trust model issues with Type II compared to Type I?
VM Security

- Isolation of VM computing
- Like a separate machine
VAX VMM Security Kernel

• A1 assured virtual machine system

• Virtualization
  • Protect sensitive state
    • Sensitive instructions must be virtualized (i.e., require privilege)
    • Access to sensitive data must be virtualized (ditto)
  • Need to hide virtualization
    • Systems cannot see that they are being virtualized
  • I/O Processing
    • Need to share access to devices correctly
    • Special driver interface (all in VMM security kernel)
  • Self-virtualization: Run VMM as VM
VM Security

- Do VMs need to communicate or share resources?
- How do they do it?
VAX VMM Access Control

- Subjects and objects
  - *Coarse-grained access control possible*
  - VMs are subjects
  - Disk partitions are objects

- Lattice policies for secrecy and integrity
  - Bell-LaPadula for secrecy
  - Biba for integrity

- Privileges for special operations
  - E.g., administrative operations

- Discretionary access controls
Aside

- **Simple security property**
  - Read-down only
  - S can read O if and only if S’s access class dominates O

- ***-security property**
  - Write-up only
  - S can write to O if and only if O’s access class dominates S

- **Basic Security Theorem**
  - Every protection state satisfies simple and *-security properties
  - Bell-LaPadula meets this trivially
  - Q: Why is this?
VAX VMM Challenges

• Q: Why was the project cancelled?
• Drivers? In VMM... New model...
• Development languages/performance? Pascal?!
• Usability? Where’s X?
• Lack of customers?
• Hardware changes?
• Covert channel defenses? Fuzzy time...
• Insanity?
NetTop

- Isolated networks of VMs
- Alternative to “air gap” security
Xen

- Paravirtualized Hypervisor
- Privileged VM
Xen sHype

- Controlled information flows among VMs

VM: DomU

Guest OS'

Partitioned Resources

VM: DomU

Guest OS'

Device Requests

Dom 0

Host OS'

Drivers

Xen Hypervisor

Ref Mon
Xen sHype Policies

- **Type Enforcement**
  - Mandatory, access matrix policy associating *subject labels* with *object labels* and *operations*
  - A VM with a subject label $L$ can perform an operation $op$ on an object (e.g., VM, memory, file system) with object label $M$ if the TE policy access matrix includes an entry for this.

- **Chinese Wall**
  - Conflict of interest restrictions
  - A subject can access an object labeled $L$ in conflict group $C$
    - If subject has previously accessed an object labeled $L$
    - If subject has not previously accessed an object of any label in conflict group $C$

- Why are Type Enforcement and Chinese Wall used?
Java Virtual Machine

- Interpret Java bytecodes
  - Machine specification defined by bytecode
  - On all architectures, run same bytecodes
    - Write once, run anywhere

- Can run multiple programs w/i JVM simultaneously
  - Different ‘classloaders’ can result in different protection domains

- How do we enforce access control?
Java Security Architecture

- Java 1.0: Applets and Applications
Java Security Architecture

- Java 1.1: Signed code (trusted remote -- think Authenticode)
- Java 1.2: Flexible access control, included in Java 2
Stack Inspection

- Authorize based on protection domains on the stack
  - Union of all sources
    - All must have permission

<table>
<thead>
<tr>
<th>class</th>
<th>method</th>
<th>protection domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.artima.security.stranger.Stranger</td>
<td>main()</td>
<td>CDROM</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend</td>
<td>doYourThing()</td>
<td>STRANGER</td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>doYourThing()</td>
<td>FRIEND</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend$1</td>
<td>dPrivileged()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>TextFileDisplayer</td>
<td>run()</td>
<td>FRIEND</td>
</tr>
<tr>
<td>java.io.FileReader</td>
<td>&lt;init&gt;()</td>
<td>CDROM</td>
</tr>
<tr>
<td>java.io.FileInputStream</td>
<td>&lt;init&gt;()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkRead()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkPermission()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>checkPermission()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.security.AccessControlContext</td>
<td>checkPermission()</td>
<td>BOOTSTRAP</td>
</tr>
</tbody>
</table>
Do Privileged

- `doPrivileged` terminates backtrace
- Like `setuid`, with similar risks

<table>
<thead>
<tr>
<th>class</th>
<th>method</th>
<th>protection domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example2b</td>
<td>main()</td>
<td>CDROM</td>
</tr>
<tr>
<td>com.artima.security.stranger.Stranger</td>
<td>doYourThing()</td>
<td>STRANGER</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend</td>
<td>doYourThing()</td>
<td>FRIEND</td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>dPrivileged()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>com.artima.security.friend.Friend$1</td>
<td>run()</td>
<td>FRIEND</td>
</tr>
<tr>
<td>TextFileDisplay</td>
<td>doYourThing()</td>
<td>CDROM</td>
</tr>
<tr>
<td>java.io.FileReader</td>
<td>&lt;init&gt;()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.io.FileInputStream</td>
<td>&lt;init&gt;()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkRead()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.lang.SecurityManager</td>
<td>checkPermission()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.security.AccessController</td>
<td>checkPermission()</td>
<td>BOOTSTRAP</td>
</tr>
<tr>
<td>java.security.AccessControlContext</td>
<td>checkPermission()</td>
<td>BOOTSTRAP</td>
</tr>
</tbody>
</table>
Virtual Machine Threats

• How does the insertion of a virtual machine layer change the threats against the system?
Virtual Machine Rootkit

• Rootkit
  – Malicious software installed by an attacker on a system
  – Enable it to run on each boot

• OS Rootkits
  – Kernel module, signal handler, ...
  – When the kernel is booted, the module is installed and intercepts user process requests, interrupts, etc.
  – E.g., keylogger

• VM Rootkit
  – Research project from Michigan and Microsoft
  – If security service runs in VM, then a rootkit in VMM can evade security
  – E.g., Can continue to run even if the system appears to be off
Take Away

• VM systems focus on isolation
  • Enable reuse, but limited by security requirements

• Enable limited communication
  • The policies are not trivial